ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TUPDP089

IMPROVING CERN'S WEB-BASED RAPID APPLICATION PLATFORM

E. Galatas*, S. Deghaye, J. Raban, C. Roderick, D. Saxena, A. Solomou
CERN, Geneva, Switzerland

Abstract

The Web-based Rapid Application Platform (WRAP) aims to provide a centralised, zero-code, drag-n-drop means of GUI creation. It was created at CERN to address the high maintenance cost of supporting multiple, often deprecated, solutions and potential duplication of effort. WRAP leverages web technologies and existing controls system infrastructure to provide a drop in solution for a range of use cases. However, providing a centralized platform to cater for diverse needs and to interact with a multitude of data sources presented performance, design, and deployment challenges. This paper describes how the architecture evolved to address technological limitations and increase usability and adoption.

INTRODUCTION

The development of CERN's Web-based Rapid Application Platform (WRAP) aims to replace the expensive task of specialised application development, and the time investment required by many people for maintenance [1]. Such a platform needs to encapsulate a great deal of complexity in its implementation, far greater than any individual user application previously developed at CERN. Focusing on WRAP's large component parts individually (including both back-end services and front-end modules), has helped navigate this challenge. This paper describes some of the more difficult challenges faced in each of these parts, together with the implemented solutions, and limitations that remain to be solved.

A summary breakdown of CERN's accelerator control system core concepts [2], in simple terms, will serve as a foundation to understanding the WRAP architecture and the aspects that it needs to address.

CONTROL SYSTEM OVERVIEW

Devices and Properties

Data produced by a device follows the so-called device-property model. Each device having one or more properties, and every property having one or more fields. For WRAP device interactions, two types of operations are possible and in most cases they are made on the property level:

1. Subscriptions to receive property updates.
2. Sets to load new values into a device property.

In some cases, operations accept a filter called "selector" that refers to a specific particle accelerator state for which the operation should be performed. A Timing system coordinates the state of the accelerators and devices react accordingly. Properties that accept such filter are marked as "multiplexed". For example, an extraction kicker magnet's strength setting depends on the extraction energy, and the property to control the strength is multiplexed to allow different settings for different beams.

The Controls Configuration Service (CCS) [3] centralizes the configuration of the entire Control System. Its CCDA (Controls Configuration Data Access) API [4] provides, amongst other things, services to retrieve the available devices, their property models and their supporting metadata and available interactions.

Subscriptions

A subscription is established directly to a device, using CMW [5], and references a property. A so-called "first update" provides an initial value when a client subscribes, which is highly valuable for device properties that update infrequently. For multiplexed properties, if a selector is provided, a first update is received for the given selector only, otherwise a value is received for each possible selector.

Data Archival

Device property data can be configured (using the CCS) to have its data logged as times series in the CERN Accelerator Logging Service (NXCALS) [6], which provides numerous facilities to process and extract logged data.

Set Operations

In general, applying settings to devices requires the user to provide values for each individual field in the property concerned, resulting in one atomic "Set" operation. CERN's setting management system "LSA" [7] supports more sophisticated interactions, including "partial sets", which allow to emit some fields from a property setting, by internally using the latest cached values. LSA also provides high-level virtual devices, with virtual properties, that can also be set and subscribed to. Behind the scenes, sets on these high-level device properties will typically modify multiple property fields of multiple underlying devices at the same time. For example, when directly controlling the high-level physics parameters of a particle accelerator, such as the tune, chromaticities, etc.

PROJECT ARCHITECTURE

The WRAP architecture can be split into four major parts:

1. **Metadata Service**: a back-end service for providing metadata.

* epameinondas.galatas@cern.ch,
stephane.deghaye@cern.ch,
jakub.raban@cern.ch,
chris.roderick@cern.ch,
dinika.saxena@cern.ch,
andreas.solomou@cern.ch

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TUPDP089

2. **Live-Data Service**: a back-end service for accessing real-time device data.

3. **Application Editor**: a front-end module for creating user applications.

4. **Application Renderer**: a front-end module for displaying an application.

METADATA SERVICE

The metadata service is responsible for providing metadata for all available data sources and for facilitating storage of user-created applications.

Providing coherent metadata can be especially challenging as there is not always a single source of truth. For example:

- The current property model of a device is always defined in the CCS. However, over time, the property model of device may change multiple times for legitimate reasons.

- Historical device data is archived in the NXCALS system, but the historical data conforms to the structure of the device properties at the time it was acquired and logged.

In addition, due to incompatibilities between programming languages or 3rd party software solutions, there are some cases where it is not easy to properly represent data. For example:

- Services written in Java lack unsigned integer support on the language level, hence there is a lack of representation of such types when modeling metadata, which gets referenced as a signed integer instead. For the same case but seen from the low-level software, the metadata type presents as unsigned.

- Similarly to above, array types in NXCALS may differ from the ones produced by the actual devices, due to incompatibilities at the type level between the programming languages and in this case, the Apache Spark framework used in NXCALS.

Without the burdens of storage and design limitations of higher level systems, the CCS would seem to be the logical source from which to obtain accurate information about the data produced by a device, however that is not always the case. High-level sets (introduced earlier) are a vital control operation that includes a validation layer of the incoming values. The metadata governing this layer can be overridden, for example, reducing the allowed value range of an integer, driven by accelerator operation needs. As such, the statically defined CCS metadata may not be accurate.

Solving this systemic issue required to develop a WRAP metadata service, which, instead of simply relaying information, reconciles metadata from multiple upstream services to provide a complete and accurate device representation to the front-end layer. This case highlights the importance and value of having clear data specifications and system semantics before tackling the implementation of any overarching system.

LIVE-DATA SERVICE

The live-data service (LDS) is responsible for subscribing to device properties and transmitting the acquired values as requested by WRAP user applications.

Caching and Aggregation

In order to reduce unnecessary load on both the devices and WRAP itself, the LDS shares subscriptions between clients. While this is essential, it breaks the aforementioned ”first update” semantics, forcing the LDS to re-implement caching of the latest property-level values. Some state management is necessary to persist and invalidate both subscription handles and data cache. In addition, two forms of value aggregation is necessary to improve semantics of end-user applications. Cached values should be aggregated:

1. By property, since properties are atomic within the control system.

2. By selector (if the property is multiplexed), to align with accelerator state.

Selector-based aggregation requires a trigger to be sent out by the Timing system, per timing domain (corresponding to the available selector values), to drive the actual aggregation process. Values that are delayed beyond a short grace period are dropped, and marked as such. This process not only avoids stale values being displayed in the applications, but also assists in detecting potential errors in the underlying data sources.

Communication Protocol

The communication between the WRAP back-end and front-end was previously done via HTTP/2 push [8], a relatively new web feature, that allows front-end requests to receive multiple responses. This was ideal for simply subscribing to a device property and receiving a stream of its relevant field values. In particular, this stream was isolated and mostly stateless, while supporting load balancing across multiple server nodes for the case of bigger applications. However, it proved detrimental when it came to implementing the aforementioned data aggregations, on top of the hard limit on the number of parallel connections allowed by web browsers.

A subsequent switch to web-socket communication opened the door to unlimited connections and led to the implementation of a custom, more flexible, use-case-specific, communication protocol. Improvements to performance were also notable, especially on very high traffic user applications, as each message does not need to be wrapped in an HTTP packet. This also improved the life of the WRAP developers as HTTP/2 push is unfortunately underused leading

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TUPDP089

to problems when developing locally, e.g. there is limited support on Mac OS.

Reactive Modelling and Performance

The overarching WRAP LDS implementation encompassing caching, synchronisation, and distribution of messages from a device connection to the front-end widgets, is modelled reactively, end-to-end, via the Observer pattern. Two major implementations of the "Reactive Extensions" specification [9] are utilised: Reactor [10] for the Java back-end and RxJS [11] for the Angular JavaScript front-end.

Further performance improvements can be made in this domain in the future, e.g. swapping message serialisation from JSON to binary format. In particular, this will benefit large (n > 1000) numeric array serialisation. Binary buffers representing arrays can also be copied trivially before rendering, and be passed by reference, if web workers were to be utilised for parallel computation. Regarding the actual encoding protocols, msgpack [12] stands out with its dynamic data schema support, and sufficient Java and JavaScript implementations.

APPLICATION EDITOR

The user-facing application editor module accounts for most of the complexity found in the WRAP front-end. Its functionalities include:

- Creating "dashboards" (application panels).
- Placing and configuring widgets.
- Modeling interactivity between widgets.
- Interacting with various data sources.

The basic layout of the editor can be seen in Fig. 1.

Tiling System

To fulfill its mission of being a rapid application platform, WRAP needs to provide its users with a powerful, yet easy to use means of dragging, dropping, and resizing widget components, within the editor, in the form of tiles hosting widgets.

Initially, an open-source library, Gridster [13], was used. However, design limitations in its API and performance issues proved insurmountable for more sophisticated applications, generating significant end-user support requests. A novel, fully custom library was created as a replacement. This new solution provides three different modes of operation to cover all potential cases:

1. A dynamic mode. This resembles the previous Gridster-based solution, providing responsive dashboard layouts that utilise the entire screen real-estate, whatever the screen size and aspect-ration.
2. A static aspect ratio mode.
3. A fixed size mode.

The latter two modes account for size-sensitive dashboard layouts e.g. when embedded graphics are utilised.

This new solution natively provides functionality to group the tiles that contain the widgets. It also has improved mouse interaction functionalities, with respect to Gridster.

Widget Properties

Modeling the widget themselves poses a challenge, as even a simple widget displaying the latest received value can be customised extensively. Text position, appearance, and coloring can be trivially expressed in CSS, while value formatting can be achieved with a small JavaScript function (the mechanics of which will be expanded upon in the following sections). However, modeling all these options, their constraints, and interactions can quickly grow in complexity. It requires extensive validation logic to guarantee that every allowed option produces a valid application. This problem cannot be avoided, however it is partially mitigated with only a limited subset of the typical customisation properties being modelled and exposed. This conservative approach focuses on having clear semantics and mitigation of breaking changes in the future. For more complex logic, such as user-defined formatting, the concept of Variables is used.

Variables

In WRAP, Variables are application-scoped, typed, virtual data sources. They can be created on demand, and they encapsulate application business logic. Example usages include:

- Propagating URL parameters to widgets: For example, a variable of type "selector" is defined, with its value present in the applications URL. This variable can in-turn, be used in place of a selector, when referencing a multiplexed data source. The result is a URL-based data filter, set on application startup.
- Altering application behaviour: For example, a numeric variable is defined, then bound to a Set widget and to multiple chart widgets as "history offset". The result is a single control (the Set widget) to adjust the rendered history range over several charts.
- Creating forms: Multiple Set widgets can be bound to a Variable as a dynamic submit event, with the Variable itself bound to a button. The result is an elegant way to produce no-code forms, especially useful when setting entire properties as discussed above.
- Facilitating scripting: A user-defined formatting function for example, can be expressed as a script accepting a data source and producing a string Variable. This can be easily displayed on a standard label widget.

While introducing Variables, effort was invested to keep additional complexity to a minimum. Almost no new semantics were introduced, as Set and Subscribe were already present for device data sources, and are now reused to manipulate Variables.

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TUPDP089

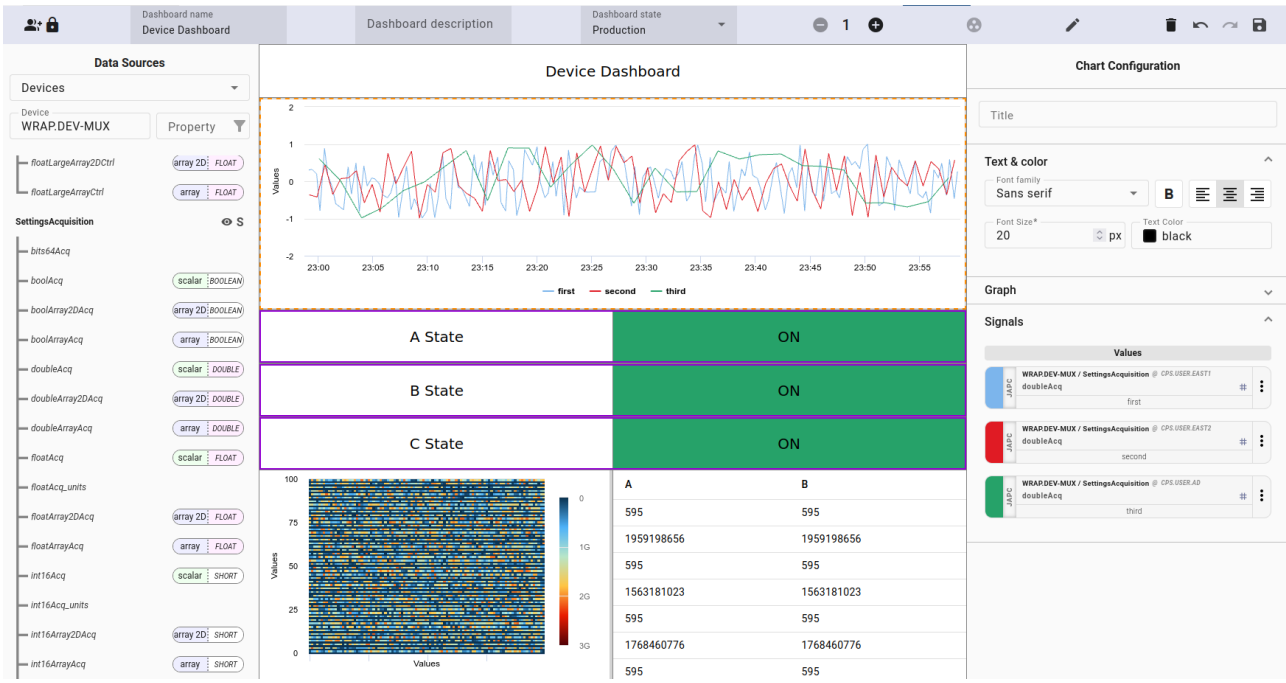

Figure 1: WRAP Application Builder.

Scripting

While WRAP is primarily a no-code platform, a significant subset of users possess the technical knowledge and have requested more advanced control possibilities. Therefore, some degree of scripting is foreseen. The easiest approach would be a custom DSL allowing very specific computations to be expressed, akin to the solution provide within Grafana [14]. Nevertheless, unconstrained scripting, e.g. evaluating JavaScript directly, exponentially raises WRAP complexity. Scripts would have to be sand-boxed in a Web Worker to avoid, at best, poor performance and, at worst, security exploits akin to an XSS attack [16]. Technical challenges notwithstanding, the WRAP application editor would have to evolve to an IDE-like solution to account for all extra debugging and validation functionality needed. For these reasons, only the DSL approach is currently being considered for implementation.

Stateful Scalable Vector Graphics (SSVG)

Expert applications sometimes need some highly-specialized visualizations. These cannot be expressed through general-purpose widgets. For such cases, it is preferable to empower the users to develop such visualizations themselves. WRAP incorporates one way of doing this, via a rather novel method, using dynamic SSVGs [15]. An SSVG is an SVG image file, extended with additional metadata that is ignored by an SVG renderer. WRAP can read this metadata and infer what data sources are accepted and how the SVG image should transform based on the state implied by in-coming WRAP data source values, such as device property updates.

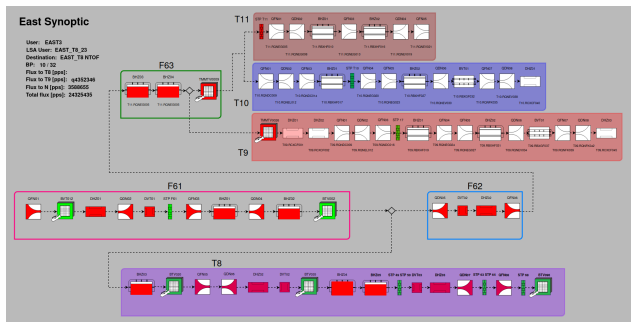

Figure 2: Synoptic panel modelled as an SSVG.

The result is dynamic visualisations defined in a platform-agnostic way. SSVGs greatly expand the scope of use cases that WRAP can handle. Examples include synoptic displays, as shown in Fig. 2, equipment visualisations (e.g. from CAD drawings), etc.

APPLICATION RENDERER

A complete application must be able to handle fast rendering speeds (fps > 30) and large data throughput (n > 1 MB/s). The main bottlenecks present in the WRAP front-end layer are the chart rendering and Angular change detection overheads.

Angular as a framework, provides tools to achieve fine grained control of updates, albeit with some code complexity penalty. Upcoming versions are expected to allow cleaner control of how each component updates, enabling incremental performance gains.

Charting performance is also paramount. A few key WRAP charting requirements include live chart updates,

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TUPDP089

support for multiple series, and high data volume visualisations. A significant performance gain was already achieved by decimating data to what can reasonably be displayed on the chart-occupied screen real estate. However, the charting library itself can have limitations. WRAP has used Highcharts [17] from the start, a library that can produce impressive visualisations with a great number of customization options. However, the live data performance does not meet the most demanding requirements and all available optimisation options have been explored. As a result, other options are now being considered. The most promising candidates are Apache ECharts [18] and uPlot [19], with the latter offering a more minimalist but high-performance solution.

CROSS-CUTTING CHALLENGES

Some challenges span the entire WRAP stack and stem from limitations of the technologies used, or they are inherent to the nature of the implemented system.

Future Evolution

Providing a comprehensive platform, with a diverse target audience, and high levels of abstraction from the underlying system introduces complexities related to its evolution. If a new feature is introduced, or deprecated, or if semantics are adjusted, an upgrade path has to be defined. During WRAP development, most major changes were preceded by a formal specification. Subsequently, an application configuration migration effort and new user documentation typically needs to be included with new WRAP versions and feature implementations, to ensure continuity of what already exists.

A thorough review process is in place at each release, incorporating feedback not only from the development team, but also from domain and underlying system experts.

64bit Integers

A major limitation that has been encountered is the lack of 64bit integer support in JavaScript and, by extension, JSON. More specifically only one number type exists stored as 64bit floating point [20] that constrains integer representation to 53 bits. Bigger integers are used across the control system to represent bit sets, serial numbers, nanosecond timestamps, and large configuration values.

Transmitting those values via JSON leads to data loss if outside the allowable range. Solutions to this problem exist, but bring significant drawbacks. String representation can provide one such solution, coupled with a custom parsing method to convert them appropriately. Another way to handle this would be a switch to a binary protocol, like the ones mentioned as options for efficient web-socket communication, but now this would be needed for simple HTTP calls as well. In all cases, added complexity is introduced due to the need for custom parsing and harder debugging, with binary formats not being readable when inspecting network traffic. Currently, there is a preference for the binary solution. Well implemented parsers exist, and will be used in other parts of the WRAP infrastructure for their performance benefits.

Assuming the transport problem is rectified, issues persist with holding and using these values on the JavaScript level. While there is a recently introduced 'BigInt' type that can fully represent them, it cannot interact with other number types, making generic processing more complex. Additionally, it actually represents arbitrary precision integers which imposes some performance penalty. Graphing libraries lack any support due to these factors and require normalisation in advance to fit in a 53 bit range.

SUMMARY

Low-code platforms that cleanly abstract complicated domains and hide the underlying technologies can bring immense value to users, improve consistency, and reduce development and maintenance costs. However, creating such systems requires underlying services to conform to clearly defined and agreed upon data semantics.

The team responsible for the platform implementation must possess multi-disciplinary knowledge to select appropriate technologies and architect the system. They must also possess or have constant access to extensive domain knowledge, to correctly model the problem space. Efforts should be made to guarantee a high degree of stability as requirements and technologies inevitably evolve, shielding the end-users from unnecessary work.

WRAP has already addressed a large number of these challenges during its implementation. It is a key pillar in CERN's controls GUI strategy, looking to extend the scope of supported use cases, leading to a much better federated GUI landscape over the next 5 years..

REFERENCES

[1] E. Galatas *et al.*, "WRAP – A Web-Based Rapid Application Development Framework for Cern's Controls Infrastructure", in *Proc. ICALEPCS'21*, Shanghai, China, Oct. 2021, pp. 894–898. doi:10.18429/JACoW-ICALEPCS2021-THPV013

[2] S. Deghaye, E. Fortescue, "Introduction to the BE-CO Control System", in *CERN Document Server*, Jan. 2020, pp 21–28. https://cds.cern.ch/record/2748122?ln=en

[3] L. Burdzanowski *et al.*, "CERN Controls Configuration Service - a challenge in usability", in *Proc. ICALEPCS'17*, Barcelona, Spain, Oct. 2017, pp. 159–165. doi:10.18429/JACoW-ICALEPCS2017-TUBPL01

[4] B. Urbaniec *et al.*, "Developing Modern High-Level Controls APIs", presented at ICALEPCS'23, Cape Town, South Africa, Oct. 2023, paper TH2AO04, this conference.

[5] J. Lauener *et al.*, "How to design & implement a modern communication Middleware based on ZeroMQ", in *Proc. ICALEPCS'17*, Barcelona, Spain, Oct. 2017. doi:10.18429/JACoW-ICALEPCS2019-WEPHA163

[6] J. Wozniak *et al.*, "NXCALS - Architecture and Challenges of the Next CERN Accelerator Logging Service",

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TUPDP089

in *Proc. ICALEPCS'19*, New York, NY, USA, Oct. 2019, pp. 1465–1469. doi:10.18429/JACoW-ICALEPCS2019-WEPHA163

[7] D. Jacquet *et al.*, "LSA- The High Level Application Software of the LHC - and its Performance During the First 3 Years of Operation", in *Proc. ICALEPCS'13*, San Francisco, CA, USA, Oct. 2013, paper THPPC058, pp. 1201–1204.

[8] M. Thomson and C. Benfield, *HTTP/2 RFC 9113, Internet Engineering Task Force*, Jun. 2022. https://www.rfc-editor.org/rfc/rfc9113

[9] Reactive Extensions, https://reactivex.io

[10] Project Reactor, https://projectreactor.io

[11] RxJS, https://rxjs.dev

[12] Msgpack, https://msgpack.org/index.html

[13] Angular Gridster 2, https://github.com/tiberiuzuld/angular-gridster2

[14] Grafana, https://grafana.com

[15] SSVG, https://mro-dev.web.cern.ch/docs/std/ssvg-specification.html

[16] XSS Attack, https://en.wikipedia.org/wiki/Cross-site_scripting

[17] Highcharts, https://www.highcharts.com

[18] ECharts, https://echarts.apache.org/en/index.html

[19] uPlot, https://github.com/leeoniya/uPlot

[20] The Number Type, ECMAScript language specification 2024, section 6.1.6.1., https://tc39.es/ecma262/#sec-ecmascript-language-types-number-type